אסטרטגיות עוורות לעומת אסטרטגיות מיודעות

- אסטרטגיות "עיוורות" אינן משתמשות בידע נוסף.

- אסטרטגיות "מיודעות" משתמשות בידע נוסף (המקודד בפונקציה יוריסטית).

חיפושים מיודעים ויוריסטיקות

- יוריסטיקה הינה טכניקה שמנצלת ידע ספציפי לתחום החיפוש כדי לזרז את החיפוש.
- פונקציה יוריסטית הינה פונקציה שמקבלת מצב ומחזירה מספר שאומד עד כמה המצב "טוב".
- בד"כ פונקציה יוריסטית מנסה להעריך את המרחק למצב הסיום (מצב בעל מרחק קטן הינו טוב יותר).
- הנושא שמייחד את הצורה בה מטפלים בתחום הבינה המלאכותית בנושא החיפוש (לעומת, למשל, תורת הגרפים) הוא השימוש בידע (יוריסטיקה) כדי לקצר את זמן החיפוש.
- חיפושים המשתמשים בפונקציה יוריסטית נקראים חיפושים מיודעים או חיפושים יוריסטיים.

דוגמא -בעית הפאזל

- יוריסטיקה: מספר המשבצות שעדיין לא במקומן.

1	2	3	4
5	6		7
13	15	11	10
9	14	12	8

ערך יוריסטי 7

1	2	3	4
5	6	11	7
13	15		10
9	14	12	8

ערך יוריסטי 8

- היוריסטיקה נותנת אינדיקציה על התקרבות למטרה

דוגמא: בעיית הניווט

- יוריסטיקה לבעית הניווט: מרחק אווירי (בהנחה שנתונות הקואורדינטות של כל מצב)
- זוהי יוריסטיקה שאנו משתמשים בה בחיי היומיום.
- במישור פתוח היוריסטיקה מדייקת לחלוטין.
- באיזור בעל מיכשולים היוריסטיקה טועה במידה רבה.

Hill-Climbing חיפוש

- בכל רגע בחיפוש נשמר צמת אחד
- אם נדרש לשחזר את המסלול אנו שומרים גם את המסלול עד אליו.
- הצומת הנוכחי מפותח. הבן המוצלח ביותר (בעל הערך היוריסטי המינימלי) שלו הופך לצומת הנוכחי (השאר נשכחים)
- דומה לחיפוש פסגה בחושך - בכל רגע בוחרים בצעד התלול ביותר.
- בגרסה אחרת מיצרים בן אחרי בן ועוברים לבן הראשון שמוצלח יותר מהצומת הנוכחי. הגרסה הראשונה נקראת Steepest-Ascent Hill Climbing הגרסה השנייה נקראת Simple Hill Climbing

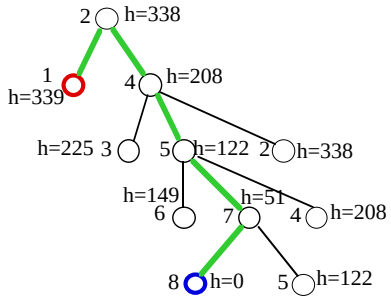
hill-climbing

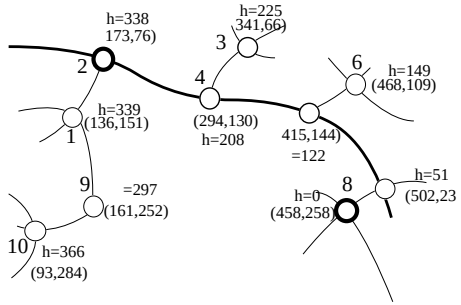
```
simple-HC (state, path)
    if goalp(state) return(path)
    else
        loop for s in succ(state)
            if h(s) < h(state) then
                simple-HC(s, path || state)
        end
        return FAIL

steepest-HC (state, path)
    if goalp(state) return(path)
    else
        let s be a state in succ(state) with maximal h
        if h(s) < h(state) then
                steepest-HC(s, path || state)
        else
            return FAIL
```


דוגמא: חיפוש במפה

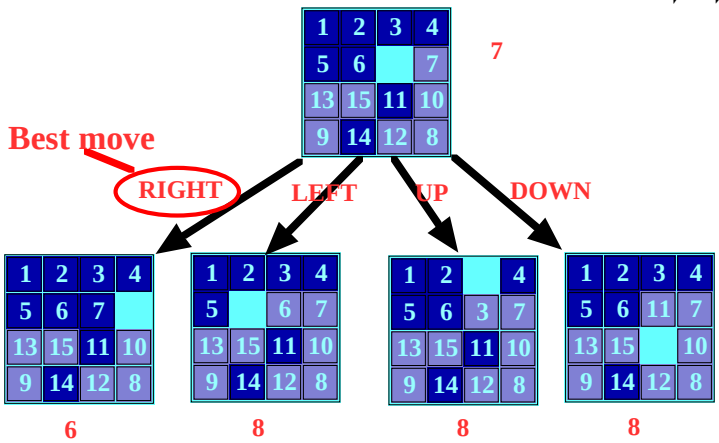
- נניח כי לכל צומת נתונות הקואורדינטות שלו.
- הפונקצייה היוריסטית מודדת את המרחק האווירי מהמטרה.
- חיפוש Hill-Climbing יצעד תמיד לצומת השכן הקרוב ביותר למטרה





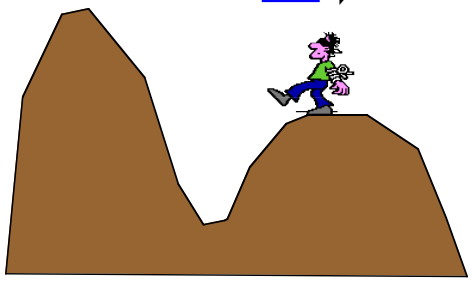
דוגמא - בעיית הפאזל

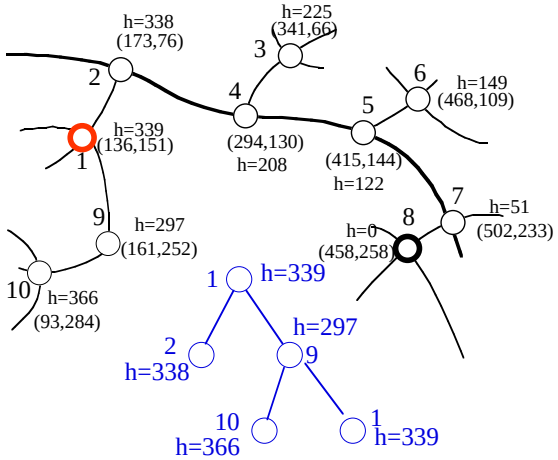
- דוגמא לצעד בחיפוש SAHC עם שימוש ביוריסטיקת "מספר המשבצות שאינן במקומן"



מינימום לוקלי

מצב שכל שכניו גרועים ממנו אך אינו מצב המטרה





פתרונות לבעית המינימום הלוקלי

- ביצוע סדרה רנדומלית (אקראית) של צעדים והמשך החיפוש מהמצב שהגענו אליו.
- ביצוע חיפוש לרוחב (או העמקה הדרגתית) מהמינימום הלוקלי עד שמגיעים למצב טוב ממנו.

חיפוש hill-climbing - ביצועים

דרישות זיכרון:

- כמספר הבנים המקסימלי לצומת (+ אורך המסלול להתחלה)
- בחיפוש simple-hill-climbing שני צמתים בלבד.
- אם עלינו לשחזר את המסלול אזי דרישות הזכרון הינו כמספר הצמתים שפותחו

יעילות חיפוש:

- תלויה לחלוטין בפונקצייה היוריסטית. פונקצייה מושלמת (שמחזירה תמיד את המרחק המדויק) תגרום לצעידה ישר למטרה. לרוע המזל אין בידינו בד"כ פונקצייה כזו.

איכות פתרון:

- שוב, תלויה בפונקצייה היוריסטית. בד"כ פתרון שאינו אופטימלי.

יתרונות של חיפוש Hill-Climbing

- אינו פועל בצורה עיוורת. משתמש בידע ספציפי (המקודד בפונקצייה היוריסטית) כדי ליעל את החיפוש.
- דרישות זיכרון נמוכות

חסרונות של חיפוש Hill-Climbing

- חיפוש לא שיטתי (מציאת המטרה אינה מובטחת גם אם קיים פתרון)
- אסטרטגיה לא הפיכה: אם התברר שהכיוון הנוכחי הוא מוטעה, אין אפשרות לחזור לנקודת החלטה קודמת.
- סכנת הקלעות למינימום לוקלי

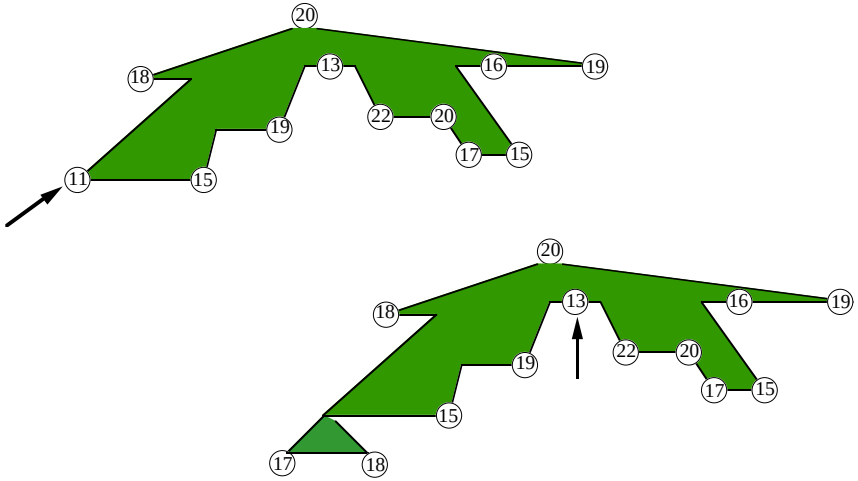
חיפוש Best-first

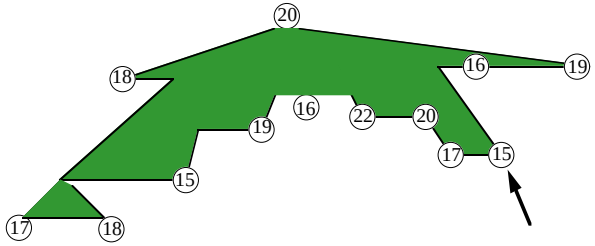
- שומר את רשימת כל הצמתים שעדיין לא פותחו ברשימה open.
- בוחר את הצומת המבטיח ביותר ב-open ומפתח אותו.
- מוסיף את רשימת הילדים לרשימת הצמתים שלא פותחו.

```
BEST-FIRST (state)
    OPEN ← (node(state,NIL,h(state)))
    While OPEN ≠ ( )
        next ← pop(OPEN)
        push (next, CLOSE )
        if goalp(next.state) then return(trace(next))
        for s in succ(next.state)
            if  s ∉ OPEN and s ∉ CLOSE then
                insert(node(s,next,h(s)),OPEN,h)
```


דוגמא לחיפוש Best-first

- המספרים מציינים את הערך היוריסטי. החץ מסמן את הצומת הבא לפיתוח.





יתרונות וחסרונות של חיפוש Best-first

יתרונות:

- חיפוש מהיר (כשהפונקצייה היוריסטית טובה)

- אינו סובל מהחסרונות של Hill-climbing

חסרונות:

- דורש זיכרון רב.
- מוצא פתרונות לא אופטימליים

חיפוש אלומה Beam-search

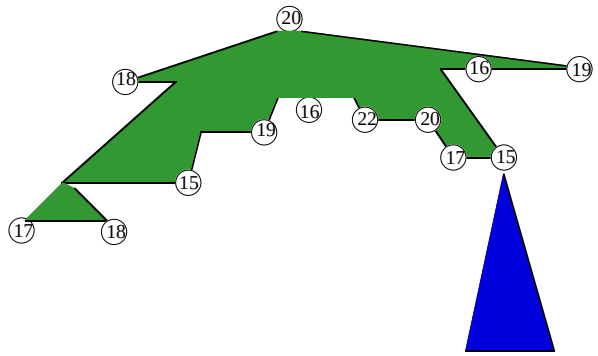
- החסרון הגדול של Best-first הן דרישות הזיכרון הגדולות שלו.
- Beam-search מגביל את מספר הצמתים שנשמרים לפיתוח לקבוע מסוים K.
- אם הוספת הילדים של צומת שפותח מביא את מספר הצמתים לערך גדול מ-K , הפרוצדורה בוחרת את ה-K הטובים ביותר (לפי הערך היוריסטי).
- ניתן לראות חיפוש Hill-climbing כחיפוש Beam-search עם K=1
- ניתן לראות חיפוש Best-first כחיפוש Beam-search עם K= ∞
- דרישות הזכרון של חיפוש אלומה עדיין גבוהות בגלל המצביעים להורים אולם פעמים רבות מבטיחה החזית הצרה התקדמות בחיפוש.


```
BEAM (state,K)
    OPEN ← (node(state,NIL,h(state)))
    While OPEN ≠ ( )
        next ← pop(OPEN)
        push (next, CLOSE )
        if goalp(next.state) then return(trace(next))
        for s in succ(next.state)
            if  s ∉ OPEN and s ∉ CLOSE then
                insert(s,OPEN,h)
                if length(OPEN)>K then
                    Delete last item of OPEN
```

דוגמא לחיפוש יוריסטי

ניסוי הדגמה: 3 סוגי חיפוש על 5 בעיות פאזל (3X3) אקראיות.

בעיה	סוג חיפוש	פותחו	זמן	אורך פתרון
1	uniform	658	25	11
	best	14	0.06	11
	hill	137	1	137
2	uniform	160	1.88	9
	best	14	0.033	11
	hill	42	0.21	לא נמצא פתרון
3	uniform	5041	1324	15
	best	129	1.4	71
	hill	300	2.66	לא נמצא פתרון
4	uniform	4618	1073	15
	best	361	9.05	109
	hill	195	1.63	לא נמצא פתרון
5	uniform	1623	143.4	13
	best	19	0.08	15
	hill	40	0.18	לא נמצא פתרון

חיפוש משולב - best-first+dfs

- הבעיה המרכזית של חיפוש best-first היא דרישת הזכרון שלו.
- חיפוש אלומה פותר את בעית הזכרון אך מקצץ ענפים העלולים להיות קריטיים.
- פתרון אלטרנטיבי - חיפוש משולב.
- מחפשים best-first עד לניצול (כמעט) כל הזכרון.
- כאשר אוזלים משאבי הזכרון מבצעים חיפוש DFS (או backtracking) מהצומת הטוב ביותר.
- כיוון שבד"כ לא ידועה הגבלת העומק, ניתן לבצע חיפוש העמקה הדרגתית.
- באם לא נמצא פתרון, מוחקים את הצומת מ OPEN ומפעילים DFS (או ID או BT) על הצומת הטוב ביותר.